Elisabeth Fughe
Matrikelnummer: 5263769
s3499227@stud.uni-frankfurt.de

Bachelorarbeit (B.Sc. - Informatik)

Können Modelle der Ökonophysik Preis-Dynamiken vorhersagen?

Fitting von Markt-Modellen auf empirischen Daten mittels Bayesscher Statistik

Elisabeth Fughe

Abgabedatum: 16. April 2019

FIAS - Frankfurt Institute for Advanced Studies
Prof. Dr. Nils Bertschinger

Zusammenfassung

tbd

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

tbd

2. Verwendete Methoden

In den nachfolgenden Kapiteln 2.1 bis 2.4 werden die mathematischen Grundlagen, die verwendeten Methoden sowie die Programmiersprache erläutert. Um verschiedene Modelle auf den gleichen Daten anzuwenden und vergleichen zu können, wurden die Modelle von Bertschinger, Mozzhorin und Sinha [1] in der Programmiersprache *Stan* beschrieben. Anschließend ist das Fitten der Modelle auf den Daten mittels **R** leicht umsetzbar.

2.1. Bayessche Modellierung

Die Bayessche Statistik untersucht mittels Bayesscher Wahrscheinlichkeiten und dem Satz von Bayes Fragestellungen der Stochastik. Anders als in der klassischen Statistik, die unendlich oft wiederholbare Zufallsexperimente voraussetzt, steht die Verwendung und Modellierung von Wahrscheinlichkeitsverteilungen im Vordergrund.
Es gilt, dass beobachtete Daten

$$x = (x_1, ..., x_n)$$

mittels bedingter Wahrscheinlichkeiten in Beziehung zu unbekannten Parametern

$$\theta = (\theta_1, ..., \theta_m)$$

stehen. So kann die gemeinsame Wahrscheinlichkeitsdichte

$$p(x, \theta) = p(x\theta) \cdot p(\theta)$$

durch die a-priori-Verteilung unbekannter Parameter $p(\theta)$ und den Erkenntnissen aus dem Datensatz $p(x|\theta)$ berechnet werden. Mit Hilfe des Satzes von Bayes kann dann die a-posteriori-Verteilung unbekannter Parameter

$$p(\theta|x) = \frac{p(x|\theta) \cdot p(\theta)}{p(x)}$$

ermittelt werden [1].

Die a-posteriori-Verteilung enthält somit Informationen über die unbekannten Parameter durch die Kombination der a-priori Verteilung mit den Informationen, die aus den beobachteten Daten gewonnen wurden. Sie wird zur Punktschätzung und zur Schätzung von Konfidenzintervallen genutzt und daher auch oft als Ziel-Verteilung bezeichnet.

So sind Bayessche Modelle im Gegensatz zur klassischen Statistik auf kleineren Datensätzen anwendbar, dort ergibt sich jedoch eine breite Wahrscheinlichkeitsverteilung, die somit unter Umständen eine geringe Genauigkeit aufweist.

2.2. Markov Chain Monte Carlo (MCMC)

In der Bayesschen Statistik beschreibt die a-posteriori-Verteilung die Unsicherheit der unbekannten Parameter, die anhand beobachteter Daten geschätzt wurden. Mit der Markov Chain Monte Carlo (MCMC) Methode können die a-posteriori-Verteilung und dadurch die unbekannten Parameter untersucht werden [8].

Um aus der Ziel-Verteilung Stichproben ziehen zu können, wird eine Markov-Kette entworfen, deren stationäre Verteilung der Ziel-Verteilung entspricht. Hier wird eine Stichprobe aus der a-posteriori-Wahrscheinlichkeitsdichte benötigt, sodass das langfristige Gleichgewicht der Markov-Kette der a-posteriori-Wahrscheinlichkeitsdichte entspricht. Der Zustand der Markov-Kette, der sich möglichst nah am Gleichgewicht befindet, entspricht dann einem Element der Stichprobe. Es gilt somit, dass sich die Qualität der Stichprobe mit der Anzahl der Schritte, die die Markov-Kette zurücklegt, verbessert.

Zur Konstruktion der Markov-Kette wird eine Übergangsfunktion berechnet, die die Ziel-Wahrscheinlichkeitsdichte invariant lässt, damit die Ziel-Dichte der stationären Verteilung der Markov-Kette entspricht. Sei z.B. $p(\theta)$ die Ziel-Dichte und sei $t(\theta^{'}|\theta)$ die Übergangsdichte, dann gilt

$$p(\theta^{'}) = \int t(\theta^{'}|\theta) \cdot p(\theta) d\theta$$

Stellt man sich die Markov-Kette als zufällig durch einen Parameterraum, also z.B. die a-posteriori-Verteilung, wandernden Punkt vor, so entspricht die n-te Komponente der Verteilung der relative Wahrscheinlichkeit, den Punkt im Zustand n anzutreffen.

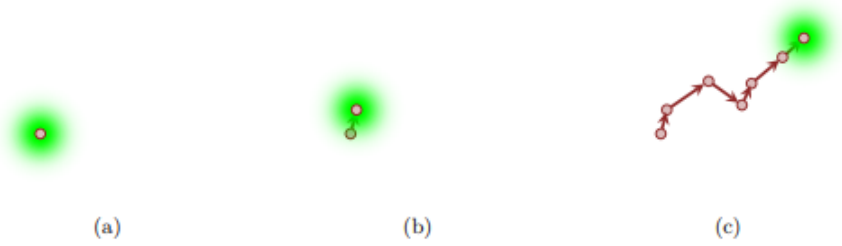

Abbildung 2.1.: Verhalten der Markov-Kette beim Ziehen von Stichproben [2]

Die Übergangsdichte (grün) (a) in Abbildung 2.1 beschreibt die Wahrscheinlichkeit eines neuen Punktes im Parameterraum in Abhängigkeit der aktuellen Position d.h. die Wahrscheinlichkeit des nächsten Schrittes der Markov-Kette auf ihrem Weg durch die Verteilung in Abhängigkeit ihrer aktuellen Position. Werden Stichproben aus dieser Verteilung gezogen, also wurden zufällig gewählte nächste Schritte vom Algorithmus akzeptiert, ergibt sich ein weiterer Zustand in der Markov-Kette und eine neue Verteilung, von der Stichproben ge-

zogen werden können (vgl. (b) in Abbildung 2.1). So wandert die Markov-Kette durch den Parameterraum (c) (vgl. Abbildung 2.1).

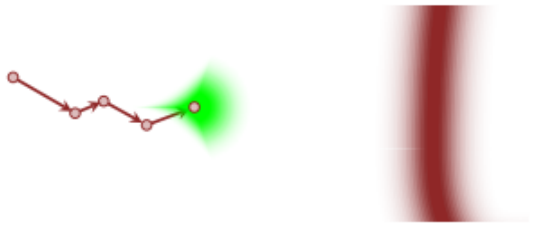

Abbildung 2.2.: Konzentration der Markov-Kette auf die typische Menge einer Verteilung [2]

Wenn die Übergangsdichte die Ziel-Verteilung beibehalten soll, konzentriert sie sich auf deren typische Menge[1] (rot) siehe Abbildung 2.2. Unter idealen Bedingungen entdeckt die Markov-Kette in drei verschiedenen Phasen:

1. Die Markov-Kette konvergiert gegen die typische Menge der Verteilung. Die MCMC-Schätzer sind stark verzerrt.

2. Die Markov-Kette hat die typische Menge erkannt und verweilt dort. Diese erste Erkundung der typischen Menge ist äußerst effektiv und verringert die Verzerrung der MCMC-Schätzer erheblich, da die Verzerrung der ersten Stichproben eliminiert wird.

3. In der dritten Phase wird die Präzision der MCMC-Schätzer durch weiteres Erforschen der typischen Menge weiter verfeinert, so dass sie den zentralen Grenzwertsatz erfüllen. Das bedeutet, dass die durch die Markov-Kette gezogenen Stichproben asymptotisch der Ziel-Verteilung folgen.

Ideale Bedingungen liegen allerdings nicht vor, wenn die typische Menge der a-posteriori-Verteilung z.B. eine starke Krümmung aufweist (siehe Abbildung 2.3). Die Markov-Kette kann damit nicht umgehen und aus diesem stark gekrümmten Bereich keine Stichprobe ziehen und ignoriert diesen einfach und verzerrt dadurch die MCMC-Schätzer. Da Markov-Ketten die exakten Erwartungen asymptotisch wiederherstellen müssen, ist eine Kompensation dafür nötig, dass manche Regionen der Ziel-Verteilung nicht beachtet wurden. Dazu bleibt die Markov-Kette an der Grenze der Krümmung hängen. So werden die MCMC-Schätzer annähernd so berechnet, als würde die Markov-Kette den gekrümmten Bereich erkunden. Allerdings erhält man so überschätzte MCMC-Schätzer. Eine starke Krümmung in der typischen Menge der Ziel-Verteilung führt also zu verzerrten MCMC-Schätzern, die den zentralen Grenzwertsatz nicht mehr erfüllen [2].

[1] **Typical Set (hier: typische Menge):** Die typische Menge ist ein Begriff aus der Informationstheorie und hängt direkt mit dem Begriff der Entropie zusammen. Es gilt, dass die Summe aller Wahrscheinlichkeiten aller Elemente der typischen Menge annähernd 1 ist [12]. Daher enthält die typische Menge eine gute Beschreibung des Parameterraums.

Abbildung 2.3.: Starke Krümmungen (grün) der typischen Menge einer Verteilung führen zu verzerrten MCMC-Schätzern [2]

Ein weit verbreitetes MCMC-Verfahren zur Konzentration auf die typische Menge der zu untersuchenden Verteilung ist der Metropolis-Hastings-Algorithmus. Der Algorithmus startet an einem zufälligen Punkt in der zu untersuchenden Verteilung. Dann wird eine Schrittweite zufällig mit Hilfe einer symmetrischen Wahrscheinlichkeitsverteilung gewählt. Der Schritt wird auf Grundlage der Wahrscheinlichkeit der neuen Position im Verhältnis zur alten Position abgelehnt oder akzeptiert [8]. So wird sicher gestellt, dass die Markov-Kette von jedem Punkt aus gegen das langfristige Gleichgewicht der Ziel-Wahrscheinlichkeitsdichte konvergiert [1].

Der Metropolis-Hastings-Algorithmus ist sehr einfach zu implementieren und liefert gute Ergebnisse, allerdings nicht bei stark korrelierten Parametern [8] und hoher Komplexität der Verteilung [2]. Denn wenn die a-posteriori-Verteilung stark gekrümmt ist, bleibt die Markov-Kette an diesen Rändern hängen und lehnt so gut wie jeden weiteren Schritt ab. So kann nicht aus allen Bereiche der Ziel-Verteilung Stichproben gezogen werden und es entsteht eine verzerrte Stichprobe.
Je mehr Dimensionen die Verteilung hat, desto kleiner wird die typische Menge der Verteilung. Das führt dazu, dass fast immer Punkte außerhalb der typischen Menge gewählt werden. Die Dichte des Punktes ist so klein, dass der Schritt vom Metropolis-Hastings-Algorithmus immer abgelehnt wird. Das führt dazu, dass sich die Markov-Kette kaum weiter bewegt. Erhöht man die Akzeptanz, indem man die Anzahl der Punkte reduziert, die sich in der typischen Menge befinden müssen, führt das dazu, dass die Markov-Kette extrem langsam konvergiert [2]. In der Theorie würde die Markov-Kette das langfristige Gleichgewicht erreichen, aber in der Praxis stehen dazu nicht unendliche Ressourcen zur Berechnung zur Verfügung.
Der Metropolis-Hastings-Algorithmus kann auch als stochastisches Optimierungsverfahren verstanden werden. Mittels Simulated Annealing[2] nähert man sich mit immer höherer Wahrscheinlichkeit an ein Minimum an. Ist die Krümmung der Verteilung zu stark, kann der Algorithmus das lokale Minimum nicht mehr verlassen und so wird fast jeder weitere Schritt der

[2] **Simulated Annealing**: „Ein heuristisches Approximationsverfahren. Es wird zum Auffinden einer Näherungslösung von Optimierungsproblemen eingesetzt, die durch ihre hohe Komplexität das vollständige Ausprobieren aller Möglichkeiten und mathematische Optimierungsverfahren ausschließen“[11].

Markov-Kette abgelehnt. Letztlich wird der Bereich der Verteilung ignoriert bzw. die Markov-Kette bleibt am Rand des unerwarteten Bereichs hängen. Das führt zu einer Verzerrung der resultierenden MCMC-Schätzer [2].

2.3. Hamiltonian Monte Carlo Sampling (HMC)

In Modellen mit vielen Dimensionen ist es möglich, dass die Markov-Kette mittels Metropolis-Hastings-Algorithmus nicht alle Bereiche der Ziel-Verteilung in einer angemessenen Zeit abtasten kann. Die Strategie des Metropolis-Algorithmus, zufällige Schritte zu wählen und danach zu entscheiden, ob der Schritt akzeptiert wird oder nicht, ist für Modelle mit vielen Dimensionen weniger erfolgreich. Denn es existieren exponentiell viele Richtungen, in die sich der Sampler bewegen kann, aber nur ein Bruchteil der Richtungen belässt den Sampler in der typischen Menge und führt letztlich zu akzeptieren Stichproben.

Um große Schritte vom initialen Punkt in unentdeckte Bereiche der Ziel-Verteilung machen zu können, müssen Informationen über die Geometrie der Verteilung genutzt werden. Die Markov-Kette soll Übergänge berechnen, die der Masse mit hohen Wahrscheinlichkeiten folgt, sich also zusammenhängend durch die typische Menge bewegen. Diese Hamiltonian-Markov-Übergänge können durch Nutzung der differenzierbaren Struktur der a-posteriori-Verteilung berechnet werden.

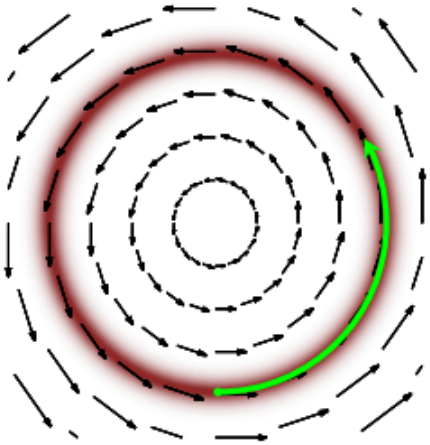

Abbildung 2.4.: Mit Hilfe eines Vektorfeldes das an der typischen Menge ausgerichtet ist, kann die Ziel-Verteilung leichter abgetastet werden [2].

Um die Zielverteilung schneller und gezielter abtasten zu können, wird jedem Punkt im Parameterraum eine Richtung für eine kleine Distanz zugewiesen, z.B. durch ein Vektorfeld, dass an der typischen Menge ausgerichtet ist (vgl. Abbildung 2.4). Folgt der Sampler den Richtungen, entsteht eine zusammenhängende Kurve (*engl. Trajectory*), die effizient und

möglichst schnell vom initialen Punkt durch unentdeckte Bereiche der typischer Menge der Ziel-Verteilung führt [2].

Um jedem Punkt eine Richtung zuzuweisen, kann ein Vektorraum mittels Informationen der Ziel-Verteilung konstruiert werden. Dazu wird die differenzielle geometrische Struktur der a-posteriori-Verteilung benötigt, die wir durch den Gradienten der Ziel-Dichte erhalten können. Denn der Gradient der Ziel-Dichte definiert eben ein solches für die geometrische Struktur der Verteilung sensitives Vektorfeld im Parameterraum.

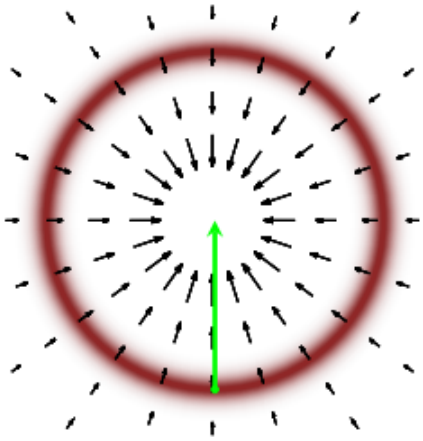

Abbildung 2.5.: Die Richtung des Gradienten zeigt immer von der typischen Menge in Richtung parameter-sensitiver Bereiche(grüner Pfeil), wie z.B. dem Modus der Verteilung. Folgt der Sampler diesen Richtungen, kann er die typische Menge nicht erkunden [2].

Allerdings hängen Ziel-Dichte sowie ihr Gradient stark von den Verteilungsparametern ab, sodass der Gradient immer in Richtung parameter-sensitiver Bereiche wie z.B. dem Modus[3] der Verteilung zeigt (vgl. Abbildung 2.5). Das Vektorfeld enthält somit nicht genügend Informationen, um den Sampler durch die typische Menge in parameter-invariante Bereiche zu führen. Um parameter-invariante Bereiche der Ziel-Dichte zu erkunden, muss der Gradient um zusätzliche geometrische Informationen ergänzt werden.

Das Ergänzen der Informationen ist durch Differentialgeometrie möglich, die auch die Mathematik der klassischen Physik beschreibt. Es gilt, dass für jedes probabilistische System ein mathematisch äquivalentes physikalisches System existiert [2]. Das Erkunden der Ziel-Dichte kann äquivalent auch als Erkunden eines physikalischen Systems beschrieben werden. Sei der Modus der Ziel-Dichte ein Planet und der Gradient sein Gravitationsfeld, dann sei die typische Menge der Dichte, der Bereich um den Planeten, durch den ein Satellit kreisen soll (vgl.

[3] **Modus**: Der Modus ist ein Begriff der desktiptiven Statistik. Er entspricht dem häufigsten Wert einer Stichprobe [10].

Abbildung 2.6).

Abbildung 2.6.: Analoges System der klassischen Physik [2].

Es gilt also das probabilistische Modell so um Impulse (*engl. Momentum*) zu erweitern, dass analog der Satellit die Anziehungskraft des Planeten ausbalanciert und die gewünschte Umlaufbahn nicht verlässt [2]. Der Hamiltonian Monte Carlo Algorithmus ist der einzige Algorithmus, der Impulse mit einer probabilistischen Struktur, die konservative Dynamiken erlaubt, implementiert [2].

Der HMC-Algorithmus zieht also Stichproben aus einem erweiterten Zustandsraum mit Impuls i eines Teilchens an der Position P.
Für die Dichte $d(P, i)$ des Zustandsraums gilt [1]:

$$\begin{aligned} d(P,i) &= d(P)d(i|P) \\ &= e^{log\ d(P)+log\ d(i|P)} \\ &= e^{-H(P,i)} \end{aligned} \tag{2.1}$$

Wobei die Hamiltonian-Korrektur $H(P, i)$ in Analogie zum physikalischen System der Summe potentieller und kinetischer Energie entspricht.

$$H(P,i) = -log\ d(P) - log\ d(i|P)$$

Die Hamiltonian Dynamiken $\hat{P}$ und $\hat{i}$

$$\begin{aligned}\hat{P} &= \frac{\partial}{\partial i} H(P, i) \\ &= -\frac{\partial}{\partial i} log\ d(i|P)\end{aligned}$$

$$\begin{aligned}\hat{i} &= -\frac{\partial}{\partial P} H(P, i) \\ &= \frac{\partial}{\partial P} log\ d(P) + \frac{\partial}{\partial P} log\ d(i|P)\end{aligned} \tag{2.2}$$

erhalten die gesamte Energie bzw. Wahrscheinlichkeit und so kann beim HMC Sampling stets ein Schritt berechnet werden, der akzeptiert wird [1], sodass die Markov-Kette auch aus stark gekrümmten Bereichen akzeptierte Stichproben ziehen kann. Besonders in Modellen mit vielen Dimensionen schafft das HMC Sampling so wichtige Effizienzvorteile.

Theoretisch ist HMC Sampling unempfindlich bezüglich stark korrelierter Parameter, allerdings werden die Differenzialgleichungen in der Praxis mittels endlicher Schrittweite gelöst. Das kann dazu führen, dass stark gekrümmte Bereiche der Ziel-Dichte doch nicht mehr erreicht werden. Außerdem gilt es die Schrittweite entsprechend der Parameter-Maßeinheiten und -Größen anzupassen [1].

2.4. *Stan* und *R*

Stan ist eine Open-Source Plattform für statistische Modellierung und high-performance Berechnungen. *Stan* ist für alle in der Datenanalyse weit verbreiteten Sprachen (R, Python, shell MATLAB, Julia, Stata) verfügbar und läuft auf den gängigen Betriebssystem (Linux, Mac, Windows) [4].

Stan ist eine höhere Programmiersprache und ermöglicht die Beschreibung der gemeinsamen Verteilung eines Modells auf einer hohen Abstraktionsebene. Die Sprache unterstützt viele Inferenz-Algorithmen, insbesondere Algorithmen, die das Gradientenabstiegsverfahren nutzen, um eine maximale a-posteriori-Schätzung zu erhalten, HMC Sampling sowie Bayessche Gradientenvariation (stochastic gradient variational Bayes). *Stan* Code wird in C++ kompiliert und die hohe Abstraktionsebene der Sprache bietet dem Nutzer einige Vorteile, z.B. müssen bei der Nutzung von HMC Sampling die Gradienten nicht manuell implementiert werden. Benötigte Gradienten werden built-in mittels C++ Library für automatisches Differenzieren berechnet [1].

Ein *Stan*-Programm muss mindestens einen `data`-, einen `Parameter`-Block sowie einen `model`-Block enthalten. Ein Beispiel dafür ist der *Stan*-Code des in dieser Arbeit verwendeten GARCH-Modells:

Im `data`-Block wird definiert, welche Daten benötigt werden, um das Modell zu fitten. Die Variablen in diesem Block entsprechen also den tatsächlichen Beobachtungen und der Block definiert, in welchem Format diese Daten vorliegen.
Hier wird zum Beispiel ein Vektor mit Zeitpunkten `T`, ein Vektor mit Angabe, ob es sich um eine tatsächliche Beobachtung oder um einen zu vorhersagenden Wert handelt, oder einen Vektor der beobachtete Returns in jedem Zeitpunkt enthält, erwartet.

Im `transformed data` Block können berechnete Daten/Variablen definiert werden. Hier zum Beispiel wird über die Daten iteriert und festgestellt, wie viele Einträge als *Missing Value* markiert wurden. So werden Beobachtungen und Vorhersagen bzw. die zu vorhersagende Daten unterschieden.

,
```
data {
  int<lower=0> T;
  int<lower=0, upper=1> miss_mask[T];
  real ret_obs[T]; // Note: Masked indices treated as missing
}
transformed data {
  int N = 0; // number of missing values
  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
```

Daran anschließend kommt der `parameters`-Block, der die unbekannten Parameter des Modells definiert. Anschließend daran kann der `transformed parameters`-Block dazu genutzt werden, um z.B. die Berechnungsvorschrift weiterer Parameter festzulegen.

Hier wird der Return für jeden Zeitpunkt `T` berechnet: Handelt es sich um eine tatsächliche Beobachtung, wird der beobachtete Return genommen, ansonsten erfolgt die Berechnung mit Hilfe der Standardabweichung im aktuellen Zeitpunkt berechnet:
`ret[t] = mu + sigma[t] * eps_miss[t]`

,
```
parameters {
  real mu;
  real<lower=0> alpha0;
  real<lower=0,upper=1> alpha1;
  real<lower=0,upper=(1-alpha1)> beta1;
  real<lower=0> sigma1;
  real eps_miss[N]; // missing normalized return innovations
}
transformed parameters {
  real ret[T]; // returns: observed or r_t=mu+sigma_t*eps_t
  real<lower=0> sigma[T];

  {
    int idx = 1; // missing value index
```

```
15
16    sigma[1] = sigma1;
17    if (miss_mask[1] == 1) {
18      ret[1] = mu + sigma[1] * eps_miss[idx];
19      idx = idx + 1;
20    } else
21      ret[1] = ret_obs[1];
22
23    for (t in 2:T) {
24      sigma[t] = sqrt(alpha0
25                      + alpha1 * pow(ret[t - 1] - mu, 2)
26                      + beta1 * pow(sigma[t - 1], 2));
27      if (miss_mask[t] == 1) {
28        ret[t] = mu + sigma[t] * eps_miss[idx];
29        idx = idx + 1;
30      } else
31        ret[t] = ret_obs[t];
32    }
33  }
34 }
```

Zum Schluss kommt der `model`-Block, der die Berechnungsvorschrift der Wahrscheinlichkeitsdichte des Modells enthält [4].
Hier die Annahme, dass `mu` und `sigma` standardnormalverteilt sind. Die Returns sind normalverteilt zu den Parametern des Modells (`mu`, `sigma`) wobei es sich, um die Standarabweichung in jedem Zeitpunkt `T` handelt. Weitere Erläuterungen befinden sich im Kapitel 3.1

Anschließend daran können im `generated quantities`-Block weitere Berechnungen definiert werden. Hier z.B. die Berechnungsvorschrift der bedingten gemeinsamen Wahrscheinlichkeitsdichte.

```
  ,
1 model {
2   mu ~ normal(0, 1);
3   sigma1 ~ normal(0, 1);
4
5   ret ~ normal(mu, sigma);
6   // Jacobian correction for transformed innovations
7   for (t in 1:T) {
8     if (miss_mask[t] == 1)
9       target += log(sigma[t]);
10  }
11 }
12 generated quantities {
13   real log_lik[T];
14
15   for (t in 1:T)
16     log_lik[t] = normal_lpdf(ret_obs[t] | mu, sigma[t]);
17 }
```

Der *Stan*-Code der Modelle, die in dieser Arbeit genutzt wurden befindet sich im Anhang und wird im Kapitel 3 näher erläutert.

Die Modelle wurden mittels *Rstan* in *R* gefittet. *Rstan* ermöglicht es *Stan*-Modelle in *R* zu kompilieren, zu testen und zu analysieren. Außerdem wurden die Pakete *tidybayes* und *tidyverse* genutzt, um die Plots zu generieren. Die tidy* Pakete zeichnen sich dadurch aus, dass sie der komplexen Datenstruktur des berechneten Fits, Informationen leicht entnehmen können und in einer Datenstruktur zur Verfügung stellen, die anschließend z.B. mittels *ggplot*-Packet leicht in *R* visualisiert werden kann.

3. Die Modelle

In den letzten Jahren haben agentenbasierte Asset-Pricing-Modelle an Bedeutung gewonnen. Im Gegensatz zur Rationalitätsannahme des Homo oeconomicus, werden nur einfache heuristische Strategien über das Verhalten von Marktteilnehmern getroffen.

Gemeinsamkeiten: readme.txt -> log returns, input, parameter, ...
stan data und parameter block beschreiben - dann pro kapitel nur die abweichenden Code lines zeigen
jacobian correction erklären
agent-based + GARCH als Vergleich

3.1. Generalized Autoregressive Conditional Heteroscedasticity (GARCH)

Generalisierte autoregressive bedingte Heteroskedastizitäts-Modelle sind stochastische Modelle zur Zeitreihenanalyse.

Autoregressive Modelle beschreiben Beobachtungen als Kombination linearer Abhängigkeit von sich selbst sowie einer Zufallsvariable z. B. zufällige Modellfehler. Die Zufallsvariable ist entsprechend nur unter Unsicherheit vorhersagbar. Mit Hilfe von autoregressiven Modellen wird also untersucht inwiefern Beobachtungen in einem Zeitpunkt von der Vergangenheit abhängen.

Autoregressive bedingte Heteroskedastizitäts-Modelle kurz ARCH-Modelle wurden erstmals von Robert F. Engle im Jahr 1982 beschrieben. ARCH(i)-Modelle treffen die Annahme, dass die bedingte Varianz der zufälligen Modellfehler im Zeitpunkt t vom realisierten Zufallsfehler der letzten i Vorperioden abhängt. Das heißt, dass große bzw. kleine Fehler dazu neigen in Clustern aufzutreten [5].

Heteroskedastizität ist ein Begriff der Statistik, der die Streuung innerhalb eines Datensatzes beschreibt. Heteroskedastizität liegt in einer Zeitreihe dann vor, wenn die Abweichung der Daten von einer Trendgeraden im Zeitverlauf zu bzw. abnimmt oder auch einfach nicht in jedem Zeitpunkt gleich ist (vgl. Abbildung 3.1). Im Gegensatz dazu bedeutet Homoskedastizität entsprechend, dass die Abweichung der Daten von der Trendgeraden in jedem Zeitpunkt gleich ist.

Eine Verallgemeinerung der ARCH-Modelle wurde 1986 von Tim Bollerslev in Form des GARCH(i,j)-Modells beschrieben. Hier hängt die bedingte Varianz der zufälligen Modellfehler im Zeitpunkt t nicht nur von Zufallsfehler der letzten i Vorperioden ab, sondern auch von sich selbst, also der bedingten Varianz der j vorherigen Zeitpunkte [3].
Sei d_t eine Zeitreihe und ϵ_t das Residuum im Zeitpunkt t, dann gilt

$$\begin{aligned} d_t &= \sigma_t \cdot \epsilon_t \\ \sigma_t^2 &= a_0 + a_1 d_{t-1}^2 + ... + a_i d_{t-p}^2 + b_1 \sigma_{t-1}^2 + ... + b_j \sigma_{t-j}^2 \end{aligned} \tag{3.1}$$

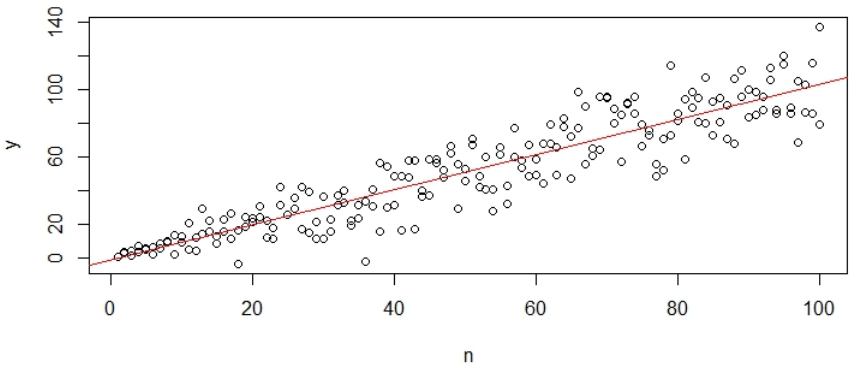

Abbildung 3.1.: Die Daten weisen im Zeitverlauf unterschiedliche Abweichung zur Trendgerade auf, d.h. die Residuen/Störterme sind nicht konstant im Zeitverlauf und es liegt Heteroskedastizität vor (Eigene Darstellung).

Wobei $a_l, b_k \in \mathbb{R}_{>0}$ mit $a_i \neq 0$ und $b_j \neq 0$ gilt. Außerdem sind die Fehlerterme unabhängige identisch verteilte Zufallsvariablen mit $E(\epsilon_t) = 0$ und $VAR(\epsilon_t) = 1$. Somit gilt, dass die bedingte Varianz

$$\sigma_t^2 = VAR(d_t|d_{t-1}, d_{t-2}, ...)$$

sowohl von Werten der eigenen Vergangenheit als auch von der Vergangenheit der gesamten Zeitreihe abhängt.

In der Ökonometrie werden autoregressive Modelle z.B. für die Modellierung und Prognose von Finanzmarktdaten genutzt. Kurse an Finanzmärkten unterliegen in der Regel keiner gleichmäßigen Volatilität. Phasen mit geringen Schwankungen alternieren mit Phasen starker Schwankungen. Die Zeitreihenanalyse und insbesondere GARCH-Modelle dienen der Identifikation und Beschreibung dieser Volatilitätscluster.

In dieser Arbeit wurde das GARCH(1,1)-Modell als Vergleichsmodell und Baseline berücksichtigt, da es ein in der Ökonometrie weitverbreitetes und genutztes Modell ist.

Beim GARCH(1,1)-Modell hängt die bedingte Varianz nur vom vorherigen Zeitpunkt ab und somit gilt

$$\sigma_t^2 = VAR(d_t|d_{t-1})$$

Entsprechend kann die GARCH(1,1) Zeitreihe wie folgt modelliert werden:

$$\begin{aligned} d_t &= \sigma_t \cdot \epsilon_t \\ \sigma_t^2 &= a_0 + a_1 d_{t-1}^2 + b_1 \sigma_{t-1}^2 \end{aligned} \tag{3.2}$$

Bei den untersuchten Zeitreihen in dieser Arbeit handelt es sich um Kursdaten des S&P 500 Indexes (vgl. Kapitel 4.1), d.h. das GARCH(1,1)-Modell untersucht Volatilitätscluster dieser Preise. Die Preise wurden in Erträge r_t umgerechnet, wobei je nach Markierung der Beobachtung der beobachtete Ertrag oder der geschätzte Ertrag r_t verwendet wird.

$$r_t = \mu + \sigma_t \cdot \epsilon_t$$

Entsprechend kann das GARCH(1,1)-Modell zur Modellierung der Volatilität in Form der Standardabweichung der Erträge implementiert werden.

$$
\begin{aligned}
\sigma_t^2 &= a_0 + a_1 d_{t-1}^2 + b_1 \sigma_{t-1}^2 \\
\sigma_t^2 &= a_0 + a_1 (\sigma_t \cdot \epsilon_t)^2 + b_1 \sigma_{t-1}^2 \\
\sigma_t^2 &= a_1 (\sigma_{t-1} \cdot \epsilon_{t-1} + \mu - \mu)^2 + b_1 \sigma_{t-1}^2 \\
\sigma_t^2 &= a_0 + a_1 (r_{t-1} - \mu)^2 + b_1 \sigma_{t-1}^2
\end{aligned} \tag{3.3}
$$

Die Berechnung wird entsprechend im *Stan*-Code als transformierter Parameter implementiert.

```
  ,
1
2  parameters {
3    real mu;
4    real<lower=0> alpha0;
5    real<lower=0,upper=1> alpha1;
6    real<lower=0,upper=(1-alpha1)> beta1;
7    real<lower=0> sigma1;
8    real eps_miss[N]; // missing normalized return innovations
9  }
10 transformed parameters {
11   real ret[T]; // returns ... observed or r_t = mu + sigma_t * eps_t
12   real<lower=0> sigma[T];
13
14   {
15     int idx = 1; // missing value index
16
17     sigma[1] = sigma1;
18     if (miss_mask[1] == 1) {
19       ret[1] = mu + sigma[1] * eps_miss[idx];
20       idx = idx + 1;
21     } else
22       ret[1] = ret_obs[1];
23
24     for (t in 2:T) {
25       sigma[t] = sqrt(alpha0
26                       + alpha1 * pow(ret[t - 1] - mu, 2)
27                       + beta1 * pow(sigma[t - 1], 2));
28       if (miss_mask[t] == 1) {
29         ret[t] = mu + sigma[t] * eps_miss[idx];
30         idx = idx + 1;
31       } else
32         ret[t] = ret_obs[t];
33     }
34   }
35 }
```

Der Erwartungswert μ sowie die Standardabweichung σ sind standardnormalverteilt. Die Erträge folgen einer Normalverteilung mit den Parametern μ und σ. Außerdem muss die Ziel-Dichte die Jacobi-Korrektur (*engl. Jacobian Correction*) für alle Vorhersagen berücksichtigen.

Schließlich wird im `model`-block des *Stan*-Codes, die a-posteriori-Dichte als normalisierte bedingte Verteilung der Erträge `ret_obs` von μ und σ definiert.

```
  ,
model {
  mu ~ normal(0, 1);
  sigma1 ~ normal(0, 1);

  ret ~ normal(mu, sigma);
  // Jacobian correction for transformed innovations
  for (t in 1:T) {
    if (miss_mask[t] == 1)
      target += log(sigma[t]);
  }
}
generated quantities {
  real log_lik[T];

  for (t in 1:T)
    log_lik[t] = normal_lpdf(ret_obs[t] | mu, sigma[t]);
}
```

Der vollständige *Stan*-Code des GARCH(1,1)-Modells befindet sich im Anhang A.1.

3.2. Franke & Westerhoff (FW)

R. Franke und F. Westerhoff haben ein Modell entwickelt, dass den Markt durch zwei Agentengruppen bzw. Handelsstrategien beschreibt: Fundamental[1]- und Chartist[2]-Trader. Händler können täglich die Strategie ändern, da im Modell ein Mechanismus modelliert wurde, der Herdenverhalten abbildet [6]
Der Anteil der Fundamental-Trader zum Zeitpunkt t sei

$$n_t^f \in [0, 1]$$

Dann gilt für den Anteil der Chartist-Trader entsprechend

$$n_t^c = 1 - n_t^f$$

da nur zwei Handelsstrategien am Markt existieren[1, 6].

[1] **Fundamental-Trader** treffen Kauf bzw. Verkaufsentscheidung auf Basis vorhandener betriebswirtschaftlicher- und Kapitalmarktinformationen wie z.B. Jahresabschlüsse, Inflation, politische Informationen etc.. Für Fundamental-Trader gilt die Annahme, dass Märkte vorhersagbar sind und auf bestimmte Vorfälle vorhersagbares Verhalten zeigen. Durch Informationen über eben diese Vorfälle, kann der Händler informierte Vorhersagen über die Preise am Finanzmarkt treffen.

[2] **Chartist-Trader** treffen Handelsentscheidungen auf Basis historischer Preise und Volumen des Marktes. Chartist-Trader analysieren graphische Repräsentationen der historischen Daten und mathematische Indikatoren, um ihre Handelsstrategie zu optimieren.

Der logarithmierte Preis p_t im Zeitpunkt t hängt vom vorherigen Preis sowie der vorherigen durchschnittlichen Nachfrage aller Händler d^f_{t-1} und d^c_{t-1} ab.

$$p_t = p_{t-1} + \mu(n^f_{t-1} \cdot d^f_{t-1} + n^c_{t-1} \cdot d^c_{t-1}) \tag{3.4}$$

Fundamental-Trader reagieren auf fehlerhafte Preise z.B. wenn der Marktpreis vom (bekannten) Fundamentalpreis[3] p^* des Vermögenswertes abweicht. Chartist-Trader reagieren im Gegensatz dazu auf Preisbewegungen der Vergangenheit z.B. $p_t - p_{t-1}$. Die Nachfrage selbst ist keine beobachtbare Größe und nur ihre gewichtete Summe beeinflusst die Preise. Sie wird von [1, 6] mittels folgender Dynamiken definiert:

$$\begin{aligned} d^f_t &= \phi(p^* - p_t) + \epsilon^f_t && \text{mit } \epsilon^f_t \sim \mathcal{N}(0, \sigma^2_f) \\ d^c_t &= \xi(p_t - p_{t-1}) + \epsilon^c_t && \text{mit } \epsilon^c_t \sim \mathcal{N}(0, \sigma^2_c) \end{aligned}$$

Allgemein gilt für zwei unabhängige Zufallsvariablen X, Y, dass die Verteilung ihrer Summe der Faltung der Verteilung der beiden Zufallsvariablen entspricht.
Sei

$$\begin{aligned} X &\sim \mathcal{N}(\mu_X, \sigma^2_X) \\ Y &\sim \mathcal{N}(\mu_Y, \sigma^2_Y) \end{aligned}$$

dann gilt

$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \sigma^2_X + \sigma^2_Y)$$

So ist es möglich, dass die Nachfrage nicht tatsächlich berechnet bzw. geschätzt werden muss, sondern indirekt durch die Verteilung der kombinierte Nachfrage der existierenden Trader-Gruppen Einfluss auf die logarithmierten Preise $r_t = p_t - p_{t-1}$ nimmt. Für das stochastische Modell der logarithmierten Preise

$$r_t \sim \mathcal{N}(\mu_t, \sigma^2_t)$$

gelten dann die Verteilungsparamter

$$\begin{aligned} \mu_t &= E[n^f_{t-1}\, d^f_{t-1} + n^c_{t-1}\, d^c_{t-1}] \\ &= \mu(n^f_{t-1}\, \phi(p^* - p_{t-1}) + n^c_{t-1}\, \xi(p_{t-1} + p_{t-2})) \end{aligned} \tag{3.5}$$

und

$$\sigma^2_t = \mu^2((n^f_{t-1})^2\sigma^2_f + (n^c_{t-1})^2\sigma^2_c) \tag{3.6}$$

Die Volatilität σ_t hängt somit von der Größe der einzelnen Trader-Gruppen ab und ändert sich mit der Zeitreihe. Um den Wechsel der Händler von einer Gruppe zur Anderen zu beschreiben, wird zu jedem Zeitpunkt t deren Anteil am Gesamtmarkt n^f_t bzw. n^c_t neu berechnet. Der

[3] Der **Fundamentalpreis** ist der Preis, der rein auf Basis vorhandener Informationen berechnet wurde

Aktualisierungs-Quotient in Formel 3.7 entspricht der DCA[4]-HPM[5] Spezifikation nach [1, 6].

Sei a_t die relative Attraktivität der Fundamental-Strategie gegenüber der Chartist-Strategie. Nach [1, 6] wird die Attraktivtät mittels drei beeinflussender Faktoren modelliert (vgl. Formel 3.8). Zum einen wird eine allgemeine Veranlagung (General Predisposition) α_0 und zum anderen Herdenverhalten (Herding) α_n und fehlerhafte Preise α_p modelliert (HPM). Da nur eine diskrete Anzahl an Wahlmöglichkeiten bestehen (DCA), ist die Wahrscheinlichkeitsdichte des Modells leicht differenzierbar. Das hat zur Folge, dass ein HMC-Algorithmus leichter Stichproben aus der a-posterior-Verteilung ziehen kann.
Wie bereits erwähnt gilt

$$n_t^c = 1 - n_t^f$$

wobei die Veränderung der Anzahl der Händler zu jedem Zeitpunkt t durch den inversen Logit[6] berechnet wird (vgl. Code 3.1 Zeile 53)

$$n_t^f = \frac{1}{1 + e^{-\beta \cdot a_{t-1}}} \qquad \beta = 1 \text{ nach [1]} \tag{3.7}$$

mit

$$a_t = \alpha_0 + \alpha_n (n_t^f - n_t^c) + \alpha_p (p^* - p_t)^2 \qquad a_n, a_p > 0 \tag{3.8}$$

Im *Stan*-Code werden die Paramter entsprechend im `parameter`- bzw. `transformed parameter`-Block definiert.

Stan-Code 3.1: Teil-Modell nach [1, 6]

```
  ,
parameters {
  real<lower=0> phi;
  real<lower=0> xi;
  real alpha_0;
  real<lower=0> alpha_n;
  real<lower=0> alpha_p;
  real<lower=0> sigma_f;
  real<lower=sigma_f> sigma_c;
  real<lower=0, upper=1> n_f_1;
  ...
  vector[N] eps_miss; // missing normalized return innovations
}
transformed parameters {
  vector[T] n_f;
  vector[T] demand;
  vector[T] sigma;
  // Note: All prices are actually log prices!
```

[4] **Discrete Choice Approach**(DCA) nach Formel 3.7 [1]

[5] **Herding**, **Predisposition**, **Missalignment**(HPM) Spezifikation für Attraktivität der Fundamental-Strategie gegenüber der Chartist-Strategie[1].

[6] Der **Logit** entspricht dem natürlichen Logarithmus der Wahrscheinlichkeit einer Chance (Quotient der Wahrscheinlichkeit p zur Gegenwahrscheinlichkeit $1 - p$). Für die Inverse gilt $p = \frac{e^X}{1+e^X} = \frac{1}{1+e^{-X}}$ [9]

```
  vector[T] p_star;
  vector[T] p;
  real ret[T];
  ...

  p[1] = 0; // wlog log p_1 = 0
  p_star[1] = ...

  n_f[1] = n_f_1;
  demand[1] = 0;

  sigma[1] = mu * sqrt( square(n_f[1] * sigma_f)
      + square((1 - n_f[1]) * sigma_c));
  {
    int idx = 1;

    if (miss_mask[1] == 1) {
      ret[1] = sigma[1] * eps_miss[idx];
      idx = idx + 1;
    } else
      ret[1] = ret_obs[1];

    for (t in 2:T) {
      // Note: index shift between prices and returns
      p[t] = p[t - 1] + ret[t - 1];
      p_star[t] = ...

      {

    // equation (HPM)
  real a = alpha_n * (n_f[t-1] - (1 - n_f[t-1]))
    + alpha_0
    + alpha_p * square(p[t-1] - p_star[t-1]);

    // equation (DCA)
  n_f[t] = inv_logit(beta * a);

    demand[t] = mu * (n_f[t] * phi * (p_star[t] - p[t])
        + (1 - n_f[t]) * xi * (p[t] - p[t-1]));

    // structured stochastic volatility
  sigma[t] = mu * sqrt( square(n_f[t] * sigma_f)
            + square((1 - n_f[t]) * sigma_c));
      }

      if (miss_mask[t] == 1) {
        ret[t] = sigma[t] * eps_miss[idx];
        idx = idx + 1;
      } else
        ret[t] = ret_obs[t];
    }
  }
```

```
}
```

Die Parameter β und μ sind redundante Parameter werden konstant gehalten, da sie nur die Größenordnung der Parameter $\alpha_0, \alpha_n, \alpha_p, f, g, \sigma_f$ und σ_c beeinflussen.
Es gilt

$$\beta = 1$$
$$\mu = 0.01$$

Die Definition der Parameter wird im *Stan*-Code im `transformed data`-Block angelegt. Also enthält dieser neben der Berechnungsvorschrift der *Missing Values* auch die Vorschrift für σ_t, μ und β.

```
  ,
transformed data {
  int N = 0; // number of missing values
  real ret_sd  = sqrt(variance(ret_obs));
  // mu and beta fixed ... redundant anyways
  real mu = 0.01;
  real beta = 1.0;

  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
```

Das Model wird anschließend im `model` bzw. `generated quantities`-Block mit finalisiert. Da wenig Wissen über die korrekte Wahl der a-priori-Verteilung für alle Parameter vorliegt, werden diese mit wenig informativen Verteilungen initialisiert, die die Skala der Parameter bestimmt. Dazu wurde die Studentsche t-Verteilung (vgl. Abbildung 3.2) gewählt, da diese schwere Ränder hat. So werden Werte, die um ein Vielfaches größer als die Standardabweichung sind, dennoch akzeptiert. Lediglich die Standardabweichungen folgen stärker informierte a-prori-Verteilungen auf Basis der Daten. Wie bereits in Kapitel 3.1 beschrieben wird auch hier die Jacobi-Korrektur modelliert.

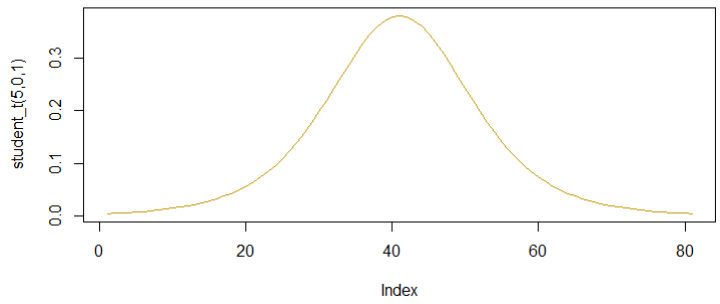

Abbildung 3.2.: Die Studentsche t-Verteilung mit 5 Freiheitsgraden, dem Erwartungswert 0 skaliert um den Faktor 1 (Eigene Darstellung).

```
  ,
model {
  phi ~ student_t(5, 0, 1);
  xi ~ student_t(5, 0, 1);
  alpha_0 ~ student_t(5, 0, 1);
  alpha_n ~ student_t(5, 0, 1);
  alpha_p ~ student_t(5, 0, 1);
  sigma_f ~ normal(0, ret_sd / mu);
  sigma_c ~ normal(0, 2.0 * ret_sd / mu);
  ...

  // Price likelihood
  ret ~ normal(demand, sigma);
  // Jacobian correction for transformed innovations
  for (t in 1:T) {
    if (miss_mask[t] == 1)
      target += log(sigma[t]);
  }
}
generated quantities {
  vector[T] log_lik;

  for (t in 1:T)
    log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
}
```

Außerdem wird angenommen, dass der Fundamentalpreis unbekannt und nicht beobachtbar ist, das wird durch den logarithmierte Fundamentalpreis $p^* = 0$ modelliert [1, 6]. Bei der Simulation des Modells auf realen Aktienmarktpreisen bzw. -erträgen, wird angenommen, dass sich der Fundamentalpreis im Zeitverlauf ändert.

Des Weiteren gilt, dass die Volatilität σ_t in Formel 3.6 über den Marktanteil der Fundamental-Trader n^f_{t-1} von der Attraktivität a_{t-2} in Formel 3.8 abhängt. Die Attraktivität wiederum wird vom logarithmierten Fundamentalpreis p^*_{t-2} beeinflusst.

In der vorliegenden Arbeit wurden zwei verschiedene Varianten des Modells nach [1] auf Marktpreisen simuliert. Die Varianten unterscheiden sich in der Berechnung der Änderungen des logarithmierten Fundamentalpreises p^*_t:

1. Modellierung von p^*_t durch die Brownsche Bewegung bzw. eine Re-Parametrierung und die Berechnung mittels zufälliger Irrfahrt (Random Walk)

2. Modellierung von p^*_t durch einen gleitenden Durchschnitt (Moving Average)

In der ersten Variante wird angenommen, dass p^*_t im Zeitverlauf der geometrischen Brownschen Bewegung[7] folgt.

$$p^* \sim \mathcal{N}(p^*_{t-1}, \sigma^2_*) \tag{3.9}$$

[7]Die **Brownsche Bewegung** ermöglicht die Simulation multiplikativer unabhängiger Zuwächse des Fundamentalpreises [7]

So enthält p_t^* durch die Brownsche Bewegung nun eine stochastische Komponente. Da p_t^* die Volatilität σ_t beeinflusst, handelt sich also um ein stochastisches Volatilitätsmodell [1, 6].

Durch eine nicht zentrierte Umparametrierung nach [1] kann die Performance des HCM-Sampler verbessert werden. Dazu wird p_t^* mittels ϵ_t^* als transformierter Parameter innerhalb einer zufälligen Irrfahrt (Random Walk) berechnet.

$$p_t^* = p_{t-1}^* + \sigma_* \cdot \epsilon_t^* \qquad \text{wobei } \epsilon_t^* \sim \mathcal{N}(0,1) \tag{3.10}$$

Das Modell bleibt dadurch unverändert, denn Formel 3.9 und 3.10 sind äquivalent [1].

Die erste Variante des Modells Θ_{FWwalk} wird also durch die Parameter ϵ_t^* und σ_* ergänzt. Im *Stan*-Code 3.2 ist diese Variante entsprechend im `parameter`- und `transformed parameter`-Block in Zeile 22 umgesetzt.

Stan-Code 3.2: Teil-Modell(Random Walk) nach [1, 6]

```
  ,
parameters {
  ...
  // p_star random walk in non-centered parameterization
  vector[T] epsilon_star;
  real<lower=0> sigma_p_star;
  ...
}
transformed parameters {
  ...
  vector[T] p_star;
  vector[T] p;
  ...

  p[1] = 0;
  p_star[1] = p[1] + epsilon_star[1];

      ...

    for (t in 2:T) {
      // Note: index shift between prices and returns
      p[t] = p[t - 1] + ret[t - 1];
      p_star[t] = p_star[t-1] + sigma_p_star * epsilon_star[t];

      {
  ...
  }
```

Der vollständige *Stan*-Code der ersten Variante befindet sich im Anhang A.2.

In Variante 2 wird die Veränderung des logarithmierten Fundamentalpreises durch einen gleitenden Durchschnitt modelliert. Es wird angenommen, dass der Fundamentalpreis von aktuellen sowie vergangenen Werten eines unbekannten stochastischen Terms (Zufallsfehler)

abhängt. Auch für diese Variante gilt somit, dass es sich um ein stochastisches Volatilitätsmodell handelt [1]. Der vollständige Stan-Code befindet sich im Anhang A.3.

random walk - treedepth 128 ?!

3.3. AL herd walk (AL)

tbd

4. Simulationen

tbd

4.1. Daten & Vorhersagen

S&P 500 data in USD from finance.yahhoo:
-daily prices - calculated into return and finally into log return as models expect log return as inputs
-exporting data with dates like Jan 1 2000 to Jan 1 2008 yahoo automatically uses last working day at stock exchange
- always used 30 Predictions - inaccurate as days per month change
=> this all leads to inaccurancy which is ok as the goal is to see the tendency in which way predictions are shifting

monthly prediction during a year in which a major crisis happend: bank crisis 2008 & dotcom 2000?
used 8 years to predict the year 2008 on a monthly basis: e.g. data from Jan 1 2000 to Jan 1 2008 used to predict Jan 2008

4.2. Ergebnisse

tbd


1
2
3
4
price
t
1.10
1.05
1.00
0.95
0.90
0.85
1800
1900
2000
0.95
0.8
0.5

Anhang

A. *Stan* Code

A.1. GARCH-Modell

```
  ,
data {
  int<lower=0> T;
  int<lower=0, upper=1> miss_mask[T];
  real ret_obs[T]; // Note: Masked indices will be treated as missing;
}
transformed data {
  int N = 0; // number of missing values
  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
parameters {
  real mu;
  real<lower=0> alpha0;
  real<lower=0,upper=1> alpha1;
  real<lower=0,upper=(1-alpha1)> beta1;
  real<lower=0> sigma1;
  real eps_miss[N]; // missing normalized return innovations
}
transformed parameters {
  real ret[T]; // returns ... observed or r_t = mu + sigma_t * eps_t
  real<lower=0> sigma[T];

  {
    int idx = 1; // missing value index

    sigma[1] = sigma1;
    if (miss_mask[1] == 1) {
      ret[1] = mu + sigma[1] * eps_miss[idx];
      idx = idx + 1;
    } else
      ret[1] = ret_obs[1];

    for (t in 2:T) {
      sigma[t] = sqrt(alpha0
                      + alpha1 * pow(ret[t - 1] - mu, 2)
                      + beta1 * pow(sigma[t - 1], 2));
      if (miss_mask[t] == 1) {
        ret[t] = mu + sigma[t] * eps_miss[idx];
        idx = idx + 1;
      } else
```

```
        ret[t] = ret_obs[t];
    }
  }
}
model {
  mu ~ normal(0, 1);
  sigma1 ~ normal(0, 1);

  ret ~ normal(mu, sigma);
  // Jacobian correction for transformed innovations
  for (t in 1:T) {
    if (miss_mask[t] == 1)
      target += log(sigma[t]);
  }
}
generated quantities {
  real log_lik[T];

  for (t in 1:T)
    log_lik[t] = normal_lpd\phi (ret_obs[t] | mu, sigma[t]);
}
```

A.2. Modell von Franke & Westerhoff (Random Walk)

```
  ,
data {
  int<lower=0> T; // time points (equally spaced)
  int<lower=0, upper=1> miss_mask[T];
  vector[T] ret_obs; // Note: Masked indices will be treated as missing;
}
transformed data {
  int N = 0; // number of missing values
  real ret_sd  = sqrt(variance(ret_obs));
  // mu and beta fixed ... redundant anyways
  real mu = 0.01;
  real beta = 1.0;

  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
parameters {
  real<lower=0> phi;
  real<lower=0> xi;
  real alpha_0;
  real<lower=0> alpha_n;
  real<lower=0> alpha_p;
  real<lower=0> sigma_f;
  real<lower=sigma_f> sigma_c;
  real<lower=0, upper=1> n_f_1;
  // p_star random walk in non-centered parameterization
  vector[T] epsilon_star;
```

```
  real<lower=0> sigma_p_star;
  vector[N] eps_miss; // missing normalized return innovations
}
transformed parameters {
  vector[T] n_f;
  vector[T] demand;
  vector[T] sigma;
  // Note: All prices are actually log prices!
  vector[T] p_star;
  vector[T] p;
  real ret[T];

  p[1] = 0; // wlog log p_1 = 0
  p_star[1] = p[1] + epsilon_star[1]; // fixme ... interpretation epsilon_raw[1]

  n_f[1] = n_f_1;
  demand[1] = 0;
  sigma[1] = mu * sqrt( square(n_f[1] * sigma_f)
      + square((1 - n_f[1]) * sigma_c));
  {
    int idx = 1;

    if (miss_mask[1] == 1) {
      ret[1] = sigma[1] * eps_miss[idx];
      idx = idx + 1;
    } else
      ret[1] = ret_obs[1];

    for (t in 2:T) {
      // Note: index shift between prices and returns
      p[t] = p[t - 1] + ret[t - 1];
      p_star[t] = p_star[t-1] + sigma_p_star * epsilon_star[t];

      {
  // equation (HPM)
  real a = alpha_n * (n_f[t-1] - (1 - n_f[t-1]))
    + alpha_0
    + alpha_p * square(p[t-1] - p_star[t-1]);
  // equation (DCA)
  n_f[t] = inv_logit(beta * a);
  demand[t] = mu * (n_f[t] * phi * (p_star[t] - p[t])
        + (1 - n_f[t]) * xi * (p[t] - p[t-1]));
  // structured stochastic volatility
  sigma[t] = mu * sqrt( square(n_f[t] * sigma_f)
            + square((1 - n_f[t]) * sigma_c));
      }

      if (miss_mask[t] == 1) {
        ret[t] = sigma[t] * eps_miss[idx];
        idx = idx + 1;
      } else
        ret[t] = ret_obs[t];
```

```
    }
  }
}
model {
  phi ~ student_t(5, 0, 1);
  xi ~ student_t(5, 0, 1);
  alpha_0 ~ student_t(5, 0, 1);
  alpha_n ~ student_t(5, 0, 1);
  alpha_p ~ student_t(5, 0, 1);
  sigma_f ~ normal(0, ret_sd / mu);
  sigma_c ~ normal(0, 2.0 * ret_sd / mu);
  epsilon_star ~ normal(0, 1);
  sigma_p_star ~ normal(0, ret_sd / 2.0);

  // Price likelihood
  ret ~ normal(demand, sigma);
  // Jacobian correction for transformed innovations
  for (t in 1:T) {
    if (miss_mask[t] == 1)
      target += log(sigma[t]);
  }
}
generated quantities {
  vector[T] log_lik;

  for (t in 1:T)
    log_lik[t] = normal_lpd\phi (ret_obs[t] | 0, sigma[t]);
}
```

A.3. Modell von Franke & Westerhoff (Moving Average)

```
  ,
data {
  int<lower=0> T; // time points (equally spaced)
  int<lower=0, upper=1> miss_mask[T];
  vector[T] ret_obs; // Note: Masked indices will be treated as missing;
}
transformed data {
  int N = 0; // number of missing values
  real ret_sd  = sqrt(variance(ret_obs));
  // mu and beta fixed ... redundant anyways
  real mu = 0.01;
  real beta = 1.0;

  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
parameters {
  real<lower=0> phi;
  real<lower=0> xi;
  real alpha_0;
```

```
  real<lower=0> alpha_n;
  real<lower=0> alpha_p;
  real<lower=0> sigma_f;
  real<lower=0> sigma_c;
  real<lower=0, upper=1> n_f_1;
  real<lower=0> lenscale_raw;
  real p_star_0;
  vector[N] eps_miss; // missing normalized return innovations
}
transformed parameters {
  vector[T] n_f;
  vector[T] demand;
  vector[T] sigma;
  // Note: All prices are actually log prices!
  vector[T] p_star;
  vector[T] p;
  real ret[T];
  real<lower=0> lenscale = 1000 * lenscale_raw;
  real<lower=0, upper=1> tau = exp( - 1 / lenscale);

  p[1] = 0; // wlog log p_1 = 0
  p_star[1] = log_mix(tau, p_star_0, p[1]);

  n_f[1] = n_f_1;
  demand[1] = 0;
  sigma[1] = mu * sqrt( square(n_f[1] * sigma_f)
      + square((1 - n_f[1]) * sigma_c));
  {
    int idx = 1;

    if (miss_mask[1] == 1) {
      ret[1] = sigma[1] * eps_miss[idx];
      idx = idx + 1;
    } else
      ret[1] = ret_obs[1];

    for (t in 2:T) {
      // Note: index shift between prices and returns
      p[t] = p[t - 1] + ret[t - 1];
      p_star[t] = log_mix(tau, p_star[t-1], p[t]);

      {
  // equation (HPM)
  real a = alpha_n * (n_f[t-1] - (1 - n_f[t-1]))
    + alpha_0
    + alpha_p * square(p[t-1] - p_star[t-1]);
  // equation (DCA)
  n_f[t] = inv_logit(beta * a);
  demand[t] = mu * (n_f[t] * phi * (p_star[t] - p[t])
        + (1 - n_f[t]) * xi * (p[t] - p[t-1]));
  // structured stochastic volatility
  sigma[t] = mu * sqrt( square(n_f[t] * sigma_f)
```

```
            + square((1 - n_f[t]) * sigma_c));
      }

      if (miss_mask[t] == 1) {
        ret[t] = sigma[t] * eps_miss[idx];
        idx = idx + 1;
      } else
        ret[t] = ret_obs[t];
    }
  }
}
model {
  phi ~ student_t(5, 0, 1);
  xi ~ student_t(5, 0, 1);
  alpha_0 ~ student_t(5, 0, 1);
  alpha_n ~ student_t(5, 0, 1);
  alpha_p ~ student_t(5, 0, 1);
  sigma_f ~ normal(0, ret_sd / mu);
  sigma_c ~ normal(0, 2.0 * ret_sd / mu);
  p_star_0 ~ normal(0, 0.2);
  lenscale_raw ~ inv_gamma(2, 1); // avoid lower boundary ... lenscale ~ inv_gamma(2, 1000)

  // Price likelihood
  ret ~ normal(demand, sigma);
  // Jacobian correction for transformed innovations
  for (t in 1:T) {
    if (miss_mask[t] == 1)
      target += log(sigma[t]);
  }
}
generated quantities {
  vector[T] log_lik;

  for (t in 1:T)
    log_lik[t] = normal_lpdf(ret_obs[t] | 0, sigma[t]);
}
```

B. *R* Code

```
  ,
data {
  int<lower=0> T;
  int<lower=0, upper=1> miss_mask[T];
  real ret_obs[T]; // Note: Masked indices treated as missing
}
transformed data {
  int N = 0; // number of missing values
  for (t in 1:T)
    if (miss_mask[t] == 1) N = N + 1;
}
```

Literatur

[1] N. Bertschinger, I. Mozzhorin und S. Sinha. „Reality-check for Econophysics: Likelihood-based fitting of physics-inspired market models to empirical data“. In: *CoRR* abs/1803.03861 (März 2018). arXiv: 1803.03861. URL: http://arxiv.org/abs/1803.03861.

[2] M. Betancourt. „A conceptual introduction to Hamiltonian Monte Carlo“. In: *arXiv preprint arXiv:1701.02434* (Jan. 2017).

[3] T. Bollerslev. „Generalized autoregressive conditional heteroskedasticity“. In: *Journal of econometrics* 31.3 (1986), S. 307–327.

[4] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li und A. Riddell. „Stan : A Probabilistic Programming Language“. In: *Journal of Statistical Software* 76.1 (Jan. 2017). DOI: 10.18637/jss.v076.i01.

[5] R. F. Engle. „Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation“. In: *Econometrica: Journal of the Econometric Society* (1982), S. 987–1007.

[6] R. Franke und F. Westerhoff. „Estimation of a structural stochastic volatility model of asset pricing“. In: *Computational Economics* 38.1 (2011), S. 53–83.

[7] *Geometrische brownsche Bewegung*. Jan. 2018. URL: https://de.wikipedia.org/wiki/Geometrische_brownsche_Bewegung.

[8] K. M. Hanson. „Markov chain Monte Carlo posterior sampling with the Hamiltonian method“. In: Bd. 4322. 2001, S. 4322 - 4322 –12. DOI: 10.1117/12.431119. URL: https://doi.org/10.1117/12.431119.

[9] *Logit*. Nov. 2018. URL: https://de.wikipedia.org/wiki/Logit.

[10] *Modus*. Juli 2018. URL: https://de.wikipedia.org/wiki/Modus_(Statistik).

[11] *Simulated Annealing*. Jan. 2019. URL: https://de.wikipedia.org/wiki/Simulated_Annealing.

[12] *Typical set*. Nov. 2018. URL: https://en.wikipedia.org/wiki/Typical_set.

Eigenständigkeitserklärung

gemäß Bachelor-Ordnung Informatik 2011 §25 Abs. 11

Hiermit erkläre ich Frau

_____

Die vorliegende Arbeit habe ich selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst.

Frankfurt am Main, den _____

_____

Unterschrift